Remarks

The amendment to claim 1

When claim 1 was amended in the response of 7/9/2008, all occurrences of "code" in the claim were to be replaced by --software object--. This was not done with the occurrence in line 12; the present amendment makes the replacement. As Examiner will immediately see, the amendment is fully supported by the Specification as filed and does not materially change the scope of the claim.

Improper final rejection

It should be pointed out at this point that Examiner cites two new references in his *Response*. Examiner does not maintain that the citation of the references is necessitated by Applicants' amendment of their claims, and if he had done so, he would not have been justified in so maintaining. Citation of new references for reasons other than a claim amendment by Applicants should not occur in final rejections; Applicants consequently respectfully request that Examine withdraw the finality of his rejection.

Traversal of the rejections under 35 U.S.C. 103

In his Office action of 4/4/2008, Examiner rejected claims 1-29 under 35 U.S.C. 103 as obvious over the combination of Monden and Valdez. Applicants' attorney believed that the failure of the combined references to disclose what Applicants are claiming in their claims was so clear that only a couple of points needed to be made; as Examiner's final rejection shows, more was necessary. Applicants will begin by rebutting Examiner's *Response* to Applicants' arguments in the response of 7/9/2008 and will then show in more detail why the references, even when combined with the newly-cited references do not disclose all of the limitations of Applicants' claims 1 and 2.

Rebuttal

What Monden and the newly-cited references disclose

In his *Response*, Examiner argues with regard to claim 1 and 22's static watermark as follows:

1. that Monden's mention of watermarking Java programs with the program's ownership, combined with the `@author` tag of Javadoc, would disclose at the very least that the program's ownership had changed.; or
2. That Taxonomy shows that Java provides the ability to watermark Java Applets with a digital signature to ensure that the actual software has not been tampered with.

Beginning with (1), the `@author` tag is used in documentation comments in Java source code which are written to be read by the Javadoc documentation generation tool. The tool "generates API documentation from documentation comments" (Javadoc, p. 4). The Javadoc tool is thus not a component of the Java execution environment and there is simply no indication in the Javadoc reference that the name of the author ever becomes part of a Java byte code file which is compiled from the source code containing the tag. It is of course the Java byte code which is executable and therefore reads onto the "software object that contains symbolic names and is executable in an execution environment" of the preamble of claim 1. Because the comments which contain the name indicated by the `@author` tag do not result in the generation of byte code, Monden's ownership watermark cannot be used by the execution environment "to determine whether the *software object* has been altered prior to the software object being executed in the program execution environment" (emphasis added), as required by claim 1.

Regarding (2), as disclosed at 5.3.1 in Taxonomy, Java Applets may be digitally signed. The digital signature is a digest of the software to be protected and is appended to the applet. In the terms used in Applicants' disclosure, such an appended digest would not be termed a watermark, since as set forth there at page 5, lines 11-13,

> A *digital watermark* is a message which has been incorporated into the content of a digital representation in such a way that the message does not render the digital representation unfit for its intended purpose.

and the digest in the Java applet is not incorporated into the Applet's content, i.e., its code.

Further, as set forth in considerable detail beginning at page 16, line 29 of Applicants' Specification, the instructions making up Applicants' static watermarks are added to the byte code at locations determined by a key which is otherwise available only to the execution environment which is executing the code. The static watermark is thus *hidden* in the byte code. There is no notion anywhere in Applicants' disclosure that their watermarks are anything other than hidden, and as is clear from the beginning of Section 2 of Taxonomy, many authorities in the digital watermarking area believe that being hidden is a fundamental part of the notion of a digital watermark. Others describe visible digital watermarks. The authors of Taxonomy deal with this issue by categorizing watermarks functionally, rather than by visibility or invisibility. It should be noted there that by their functional classification at 5.3.1, the digital signature of an Applet is a Validation Mark. Applicants' static watermark is of course another example of a Validation Mark. The last paragraph of 5.3.1 then states,

> We expect Validation Marks to be fragile and visible. The end-user must be able to detect a Validation mark to be assured of the integrity of the underlying object, and an appropriate level of fragility of a Validation mark provides assurance that the object hasn't been modified to the point that it has become invalid.

Applicants' static watermarks are of course hidden, and Taxonomy thus not only does not disclose Applicants' static watermarks, but also effectively teaches away from them.

The foregoing makes it abundantly clear that even when Taxonomy and Javadoc are added to Monden and Valdez, the combined references still do not disclose Applicants' static watermarks. Though Applicants' disclosure makes it abundantly clear that Applicants' static watermarks are invisible watermarks, Applicants are of course ready by way of Examiner's amendment or otherwise to set forth in their claim 1 that their static watermarks are hidden.

<u>Rebuttal of Examiner's contention that Valdez uses encryption to obfuscate symbolic names</u>

Examiner continues to rely on Valdez for the portions of Applicants' claim 1 that involve obfuscation and encryption. The fundamental problem with the rejection is that

Applicants are not claiming "encryption ... as a way to obfuscate code" (final Office action, page 4). The relevant language of Applicants' claim 1 as presently amended is the following:

> one or more obfuscated symbolic names that correspond to system symbolic names;
>
> a first association between the obfuscated symbolic names and encrypted forms of the corresponding system symbolic names; ...
>
> the execution environment including a second association of the encrypted forms with information needed to resolve the corresponding system symbolic names and the execution environment using the first and second associations to resolve the obfuscated symbolic names

The above is not substantially modified from the equivalent language of claim 1 as originally filed. The equivalent language was cited in the *Prosecution of the PCT application* section of the *Summary of the Prosecution* of the present response. This section was also included in the response to the first Office action in the US National Phase.

What the claim limitations set forth above require is the following:

- that the executable software object of the claim include "a first association between the obfuscated symbolic names and encrypted forms of the corresponding system symbolic names" and
- that the execution environment of the claim include "a second association of the encrypted forms with information needed to resolve the corresponding system symbolic names"

About the only disclosure of Valdez which is relevant to the limitations set forth above is that both encryption and obfuscation can be used to complicate the reverse engineering of code. The environment in which Valdez applies encryption and obfuscation to code is, however completely different from the "executable software object" and "execution environment" of Applicants' claim and the code to which the encryption is applied has nothing like the "system symbolic names" of Applicants' claim.

Beginning with the environment, as set forth at Section 6 of the reference, Valdez applies his techniques to Tcl (Tool Command language), which "is a string-based language that has a simple syntax. Tcl programs are scripts which are parsed and interpreted by a Tcl interpreter." The transformations which Valdez applies to a Tcl program are applied "on the script level" (end of Section 6). In reading the reference onto the claim, the Tcl script must be taken to be the "software object" and the Tcl interpreter must be taken to be the "execution environment".

To produce a "hidden program", Valdez applies obfuscating transformations and scrambling transformations to the Tcl script; the transformations are shown in Valdez' Table 1. The obfuscating transformations are:

- deletion of variable names after last use;
- inserting random computations and dead code
- removing comments and white space and replacing user-defined variable and procedure names with other strings;
- replacing the newline character with ";"
- appending a ";" character to a block.

The scrambling transformations are:

- partitioning a script into blocks
- transposing the positions of blocks
- compressing randomly-chosen blocks
- encrypting randomly-picked blocks
- Converting strings into hex64 format
- XOR ing two hex64 strings and returning a hex64 string.

Two things should be noted about the above list:

1. the only variable and procedure names which are obfuscated by replacing them with other strings are user-defined. There is thus no obfuscation of system symbolic names as those are defined in Applicants' Specification at page 7, line 5: "system symbolic names [that are] defined in the execution environment".
2. Encryption is applied to randomly-picked blocks of code, not to syntactic elements in the code.

The process of producing a hidden Tcl program is set forth at 5.1. The transformations are performed by a hiding tool. The inputs to the hiding tool are the Tcl script to be transformed and a profile that specifies the sections of the script to which the transformations are to be applied and parameters for the transformation including the types of transformations to be applied and the degree of transformation. As disclosed in the reference, the selected obfuscating translations are applied first; then the selected scrambling translations are applied. As the hiding tool performs the translations, it saves the information it needs to reverse each translation and then processes that information to produce an execution credential.

5.2 explains how the hidden Tcl program is executed: The execution credential and the hidden Tcl program are input to a descrambling routine, which uses the information in the execution credential to reverse the transformations. The result is the original Tcl program, which is then executed by the interpreter. The descrambling routine may be a component of the operating system, may be part of a shared library, or may be appended to the hidden program.

If Valdez is to be read onto claim 1, the hidden Tcl program must be taken to be the software object of the claim 1. When the hidden Tcl program is taken as the software object, it will be immediately seen that the hidden Tcl program has no "obfuscated symbolic names that correspond to system symbolic names", as required by the claim; further that because encryption is applied to randomly-selected blocks of code and not to individual syntactic elements, there can be no "first association between the obfuscated symbolic names and encrypted forms of the corresponding system semantic names".

Continuing with the reading, as already set forth, the Tcl interpreter must be taken to be claim 1's execution environment. The interpreter, either singly or in combination with the execution credential and the descrambling routine has no "second association of the encrypted forms with information needed to resolve the corresponding system symbolic names", as required by the claim because Valdez' encryption is applied to random blocks of the Tcl script, not to syntactic elements such as names, so there are no "encrypted

forms of the corresponding system names" as required by the claim. Further, because there is neither a "first association between the obfuscated symbolic names and encrypted forms of the corresponding system symbolic names" nor a "second association of the encrypted forms with information needed to resolve the corresponding system symbolic names", the Tcl interpreter, either singly or in combination with the execution credential and the descrambling routine, cannot resolve the obfuscated system symbolic names as set forth in the last clause of the claim.

Failure of the combination of Valdez, Monden, and the newly-cited references to render claim 1 obvious

Because neither Valdez nor Monden together with the newly-cited references disclose claim 1's static watermark that has been added to the software object or claim 1's first association between obfuscated symbolic names and encrypted forms of the corresponding system symbolic names and second association of the encrypted forms with information needed to resolved the corresponding system symbolic names, Examiner's rejection of claim 1 as obvious over the combination of Valdez and Monden is without foundation.

As Examiner will immediately see, the same logic applies also with regard to claim 22's first association between obfuscated symbolic names and encrypted forms of the corresponding system symbolic names and second association of the encrypted forms with information needed to resolved the corresponding system symbolic names. The references also fail to disclose claim 22's

> adding a method to the software object that determines whether the software object has been modified by the host;
>
> using the first and second associations to resolve the obfuscated symbolic names; and
>
> executing the added method to determine whether the software object has been modified by the host.

Examiner cites Monden, 4.1, Phase 1 with regard to this limitation. The problem with Examiner's reading is that, Monden hides his ownership watermark in a dummy method which is added to the Java source program, whereas applicants' claim 22 requires that the

method of the claims be added "during execution of the software object in the execution environment". What is involved here is not a mere implementation detail. *Because* the "method ... which determines whether the program has been modified" is added as part of *program execution* in the host, it is possible to detect whether *the host* has modified the program *prior to* its execution.

The claims dependent from claims 1 and 22 are of course all patentable because they are dependent from patentable claims, but it should also be pointed out that dependent claims 2 and 5 contain limitations that are not disclosed in he references and are therefore are patentable in their own rights over the references and that the same is the case with dependent claims 24-29.

The effect of the use of the Java programming environment and virtual machine to implement an invention on patentability

In his rejection of claims 1 and 22, Examiner raises a general issue which Applicants will deal with here:

> In essence, Applicant has attempted to distinguish the instant invention over the prior art by appealing to functionality inherent to the Java execution environment and arguing that using said functionality in the manner for which it was intended would result in a novel and non-obvious invention. Examiner strongly disagrees, noting that if applying the techniques disclosed by Monden (in view of Valdez) on a Java program that already had authorship attribution and/or digitally signing an obfuscated Java program would lead to success, then the invention is not the product of innovation but of ordinary skill and common sense. *KSR v. Teleflex*, 550 U.S. at _, 82 USPQ2d at 1397

(Final Office action, page 3)

The first problem with this argument is that, as set forth in great detail above, Monden, the newly-cited references, and Valdez do not disclose what Applicants are claiming. The second problem is that what Examiner is arguing here is that using facilities of the Java programming environment and virtual machine to implement inventions involving watermarks and/or digests and transformations involving encryption and obfuscation is

"not the product of innovation but of ordinary skill and common sense". Like any other implementation techniques, those provided by the Java programming environment and virtual machine may be used to implement obvious inventions, but the fact that the Java programming environment and virtual machine are used to implement an invention does not *by itself* establish that the invention "is not the product of innovation but of ordinary skill and common sense."

The rejections of claims 6-21

These claims are addressed to class loaders which may be used in the execution of the software object of claim 1 and the method of protecting a software object of claim 22. The following limitations from independent claim 6 appear also in claim 14:

> the improved class loader extends the class on execution of the program in the program execution environment by
> using the first association and a second association between the encrypted forms and information used to resolve the symbolic names defined in the class to resolve the symbolic names in the program, and
> adding a method to the program which determines whether the program has been modified by the host

As already pointed out in the discussion of claims 1 and 22 above, none of these limitations is disclosed in the references cited thus far.

Because the cited references do not disclose the above limitations, Examiner's rejection of claims 6 and 14 as obvious over the references is without basis and the claims are patentable over the references. Claims 7-13 and 15-21 are dependent on claims 6 and 14 respectively, and are consequently also patentable over the references. However, as Examiner will immediately see, each of the dependent claims 7-13 and 15-21 includes one or more further limitations which are not disclosed in the references, and the dependent claims are consequently also patentable in their own rights over the references.

Conclusion

Applicants have amended claim 1 to correct an error therein and have demonstrated that the amendment is fully supported by the Specification as filed and does not materially

affect the scope of the claim. Applicants respectfully request that Examiner enter the amendment as permitted by 37 C.F.R. 1.116. Applicants have further again traversed all of Examiner's rejections of their claims. Given the Examiner's use of new references in his final rejection and the complete failure of the references cited thus far to disclose anything like what is being claimed, Applicants respectfully request that Examiner withdraw the finality of his rejection, make a new search, and continue with his examination. Failing that, Applicants respectfully request that Examiner allow the claims as amended in this response. No fees are believed to be required for this amendment. Should any be, please charge them to deposit account number 501315.

Respectfully submitted,

/Gordon E. Nelson/
Attorney of record,
Gordon E. Nelson
57 Central St., P.O. Box 782
Rowley, MA, 01969,
Registration number 30,093
Voice: (978) 948-7632
Fax: (866)-723-0359
12/24/2008
Date